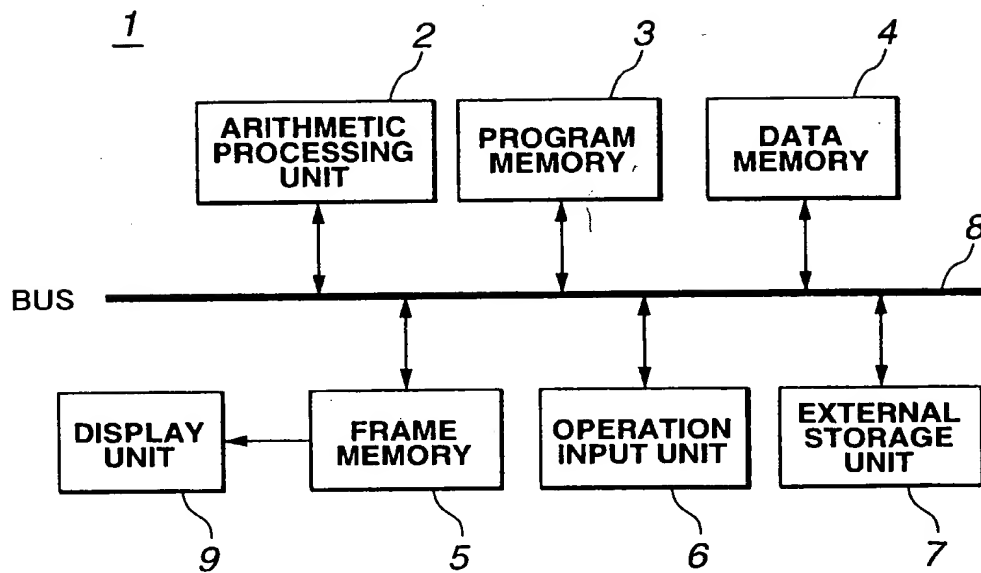
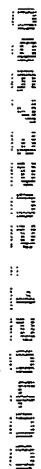


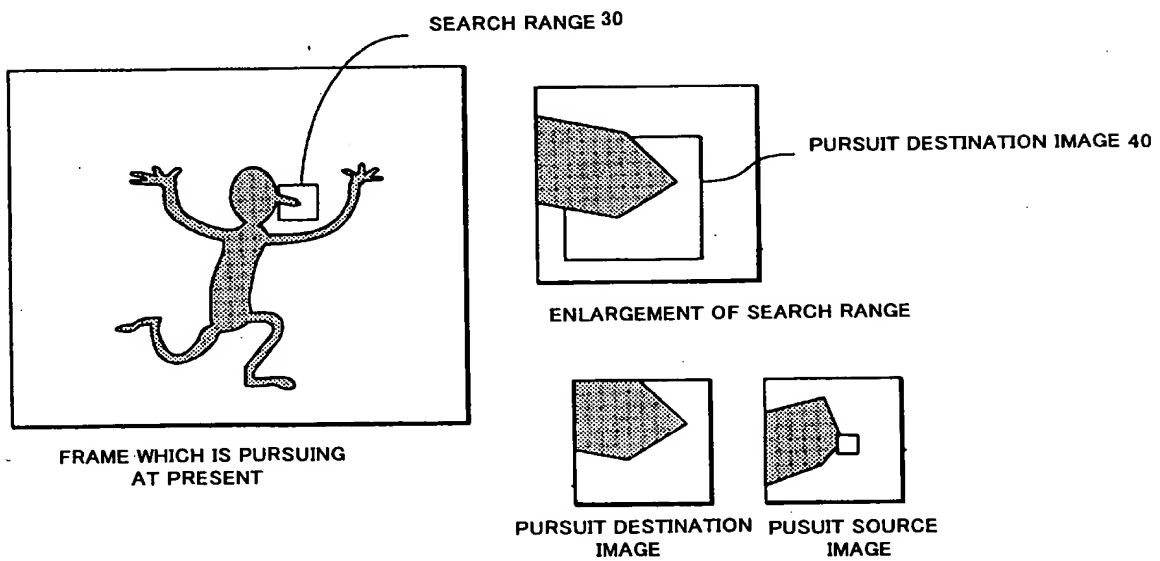
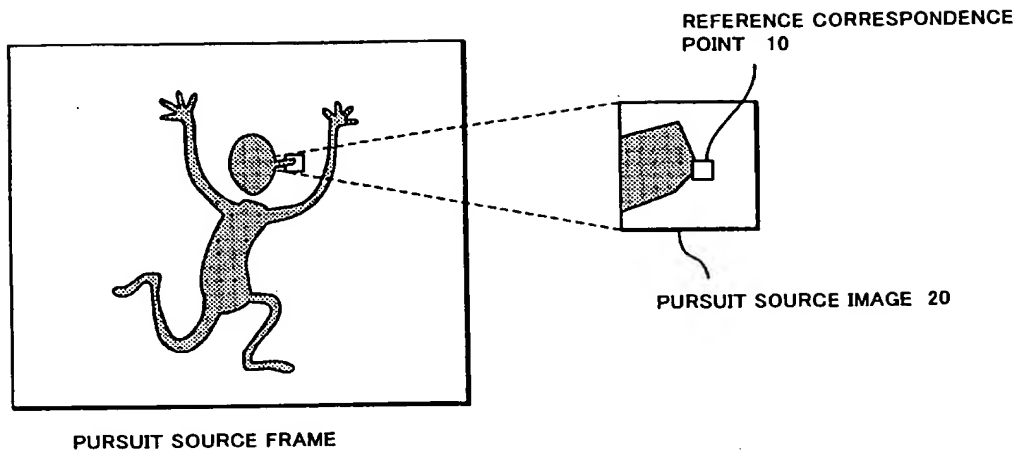
[FIG. 1]



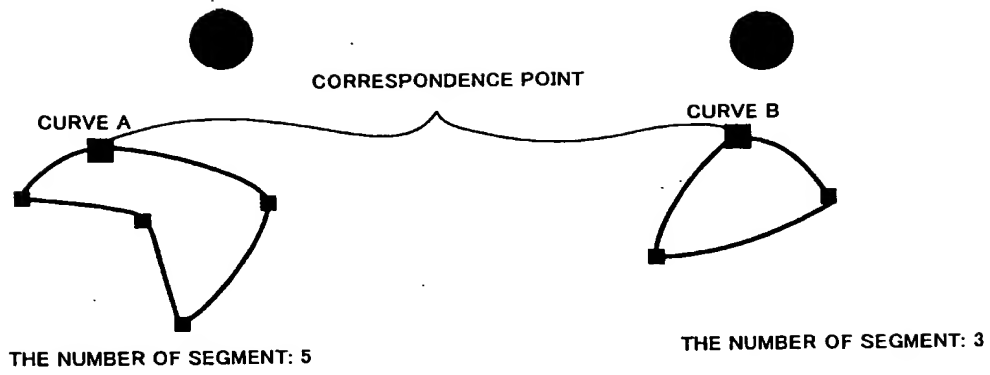




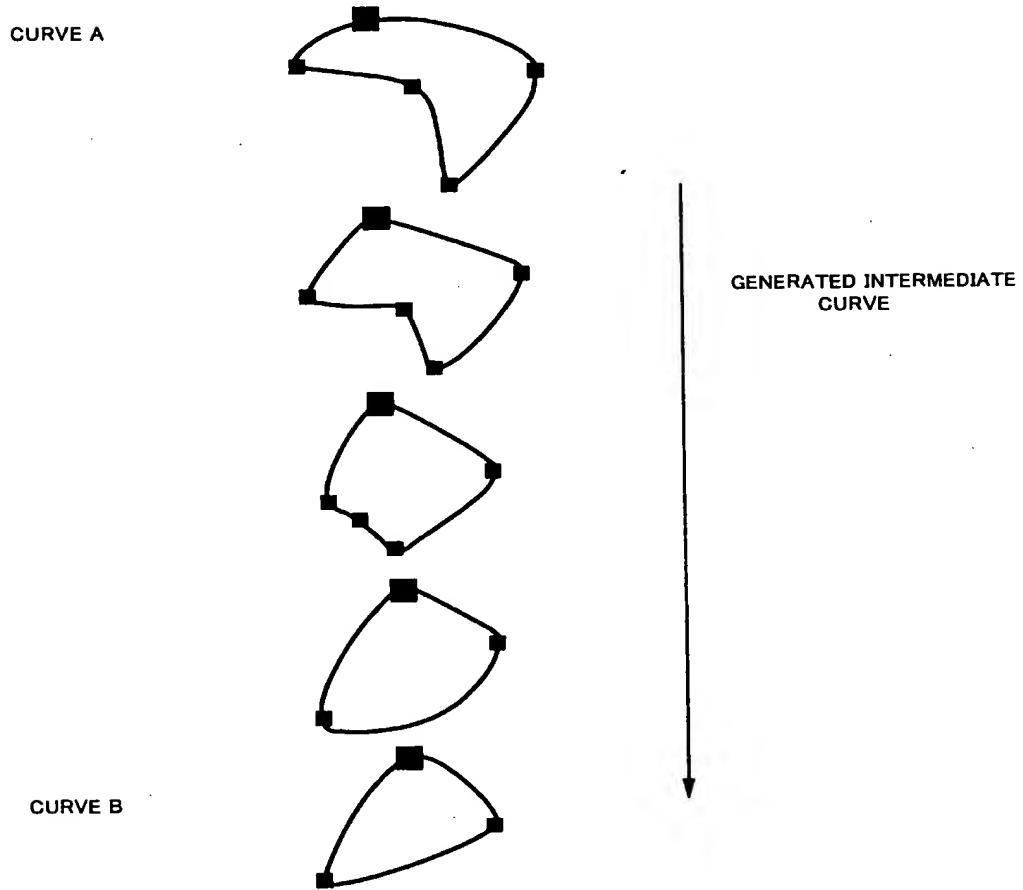
[FIG. 4]



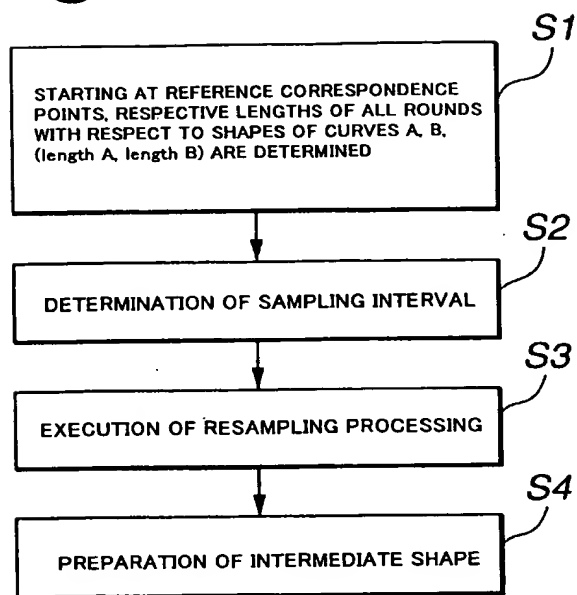
[FIG. 5]



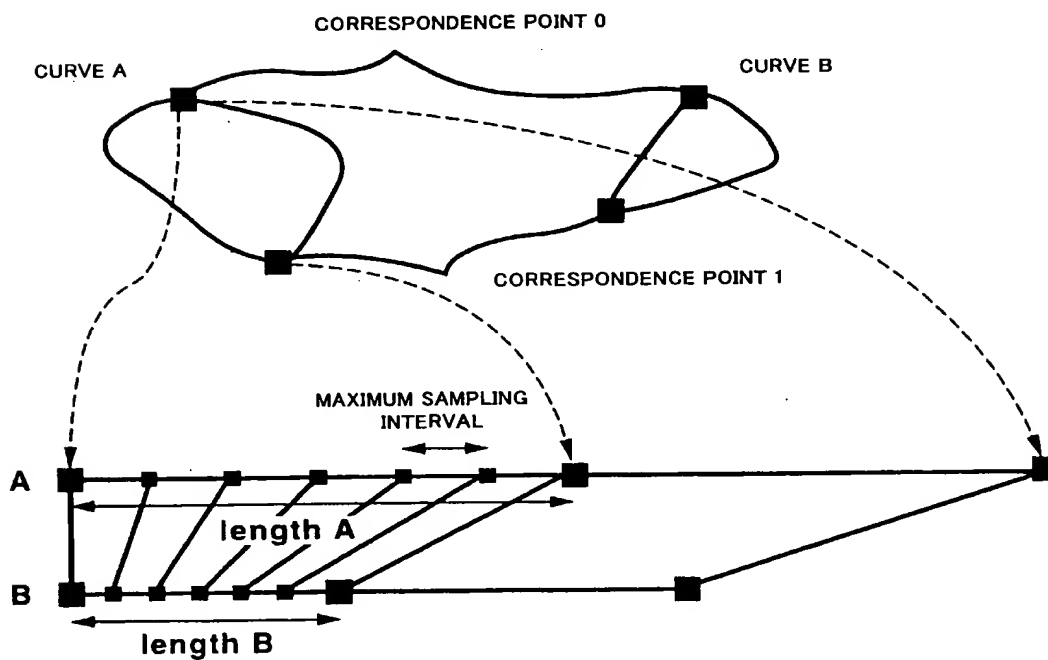
[FIG. 6]



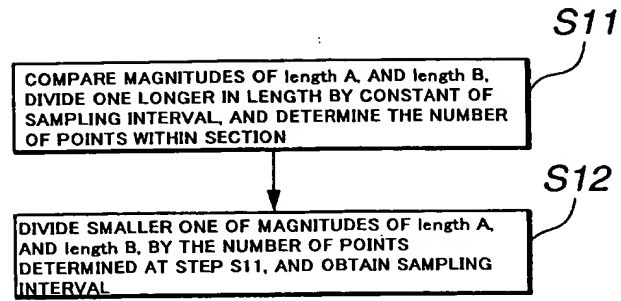
[FIG. 7]



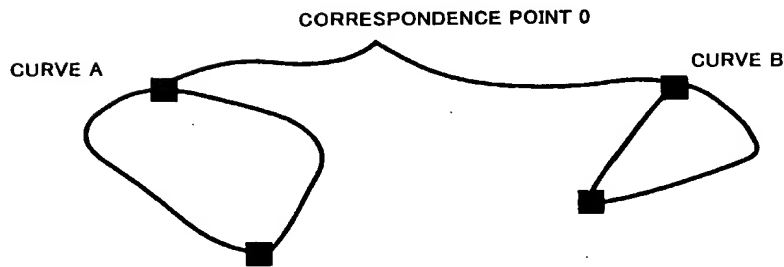
[FIG. 8]



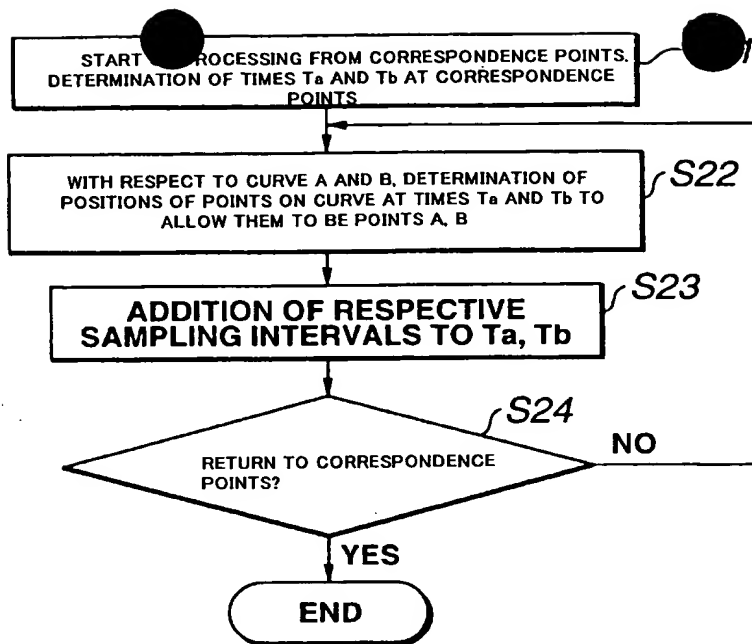
[FIG. 9]



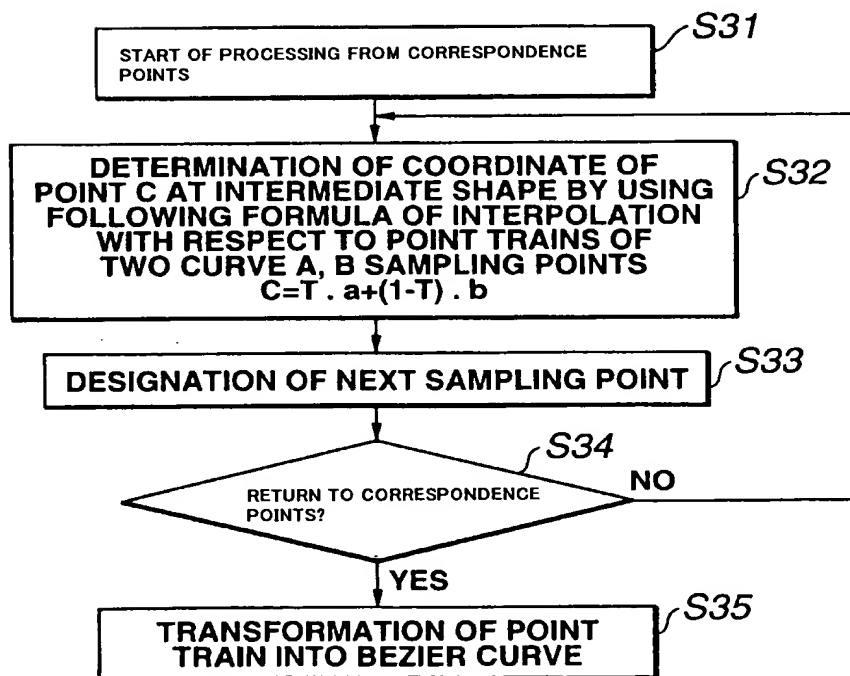
[FIG. 10]



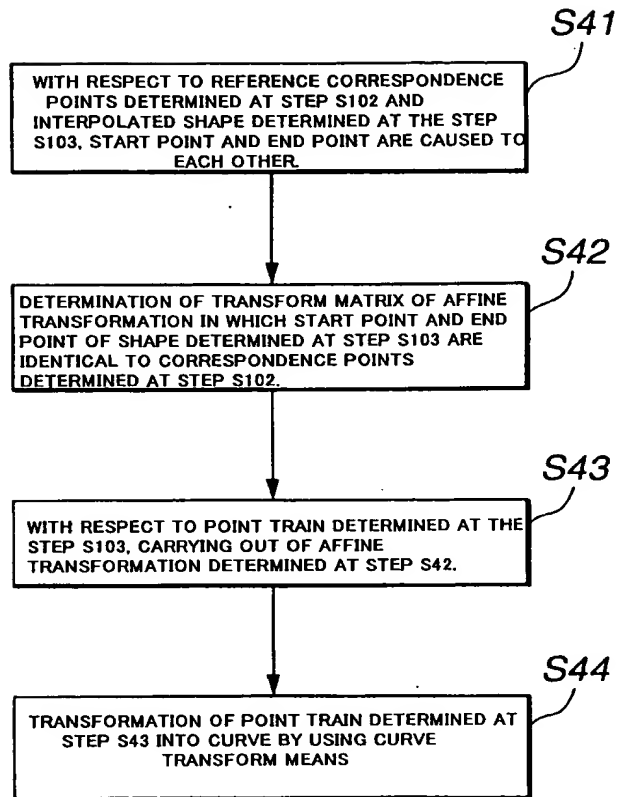
[FIG. 11]



[FIG. 12]



[FIG. 13]

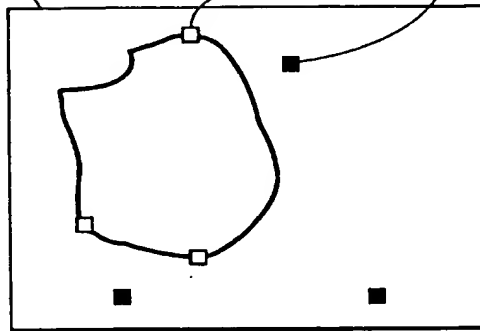


[FIG. 14]

START

SHAPE PREPARED BY INTERPOLATION 100

POINT TRAIN DETERMINED BY IMAGE PURSUIT 110

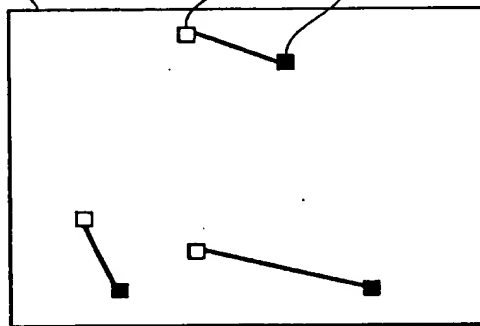


SHAPE PREPARED BY INTERPOLATION AND  
POINT TRAIN DETERMINED BY IMAGE PURSUIT

S41

START POINT 115

END POINT 116

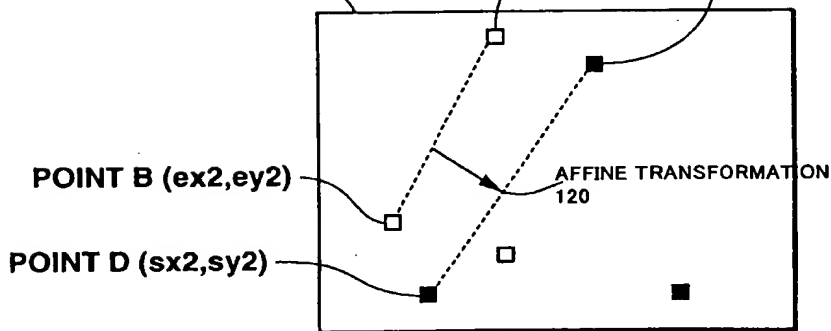


CORRESPONDENCE OF REFERENCE  
CORRESPONDENCE POINT

S42

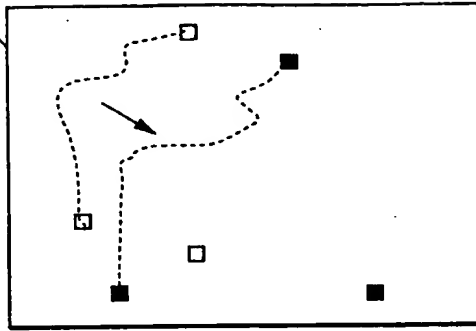
POINT A ( $ex1, ey1$ )

POINT C ( $sx1, sy1$ )



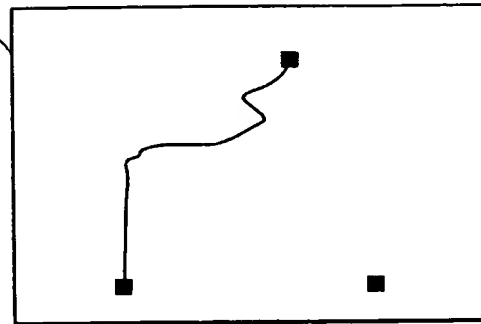
DETERMINE AFFINE TRANSFORMATION WHICH  
IS DEFORMED AS DESCRIBED ABOVE WITH  
RESPECT TO CORRESPONDENCE SECTION.

**S43**



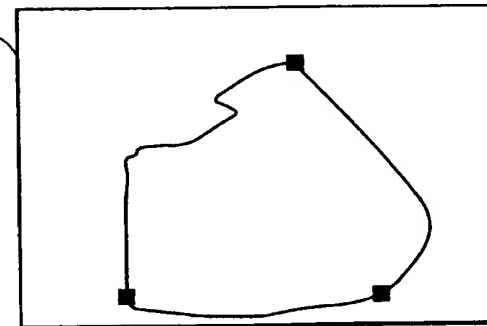
**TRANSFORMING WITH RESPECT TO POINT TRAIN  
CONSTITUTING INTERMEDIATE SHAPE AS WELL**

S44

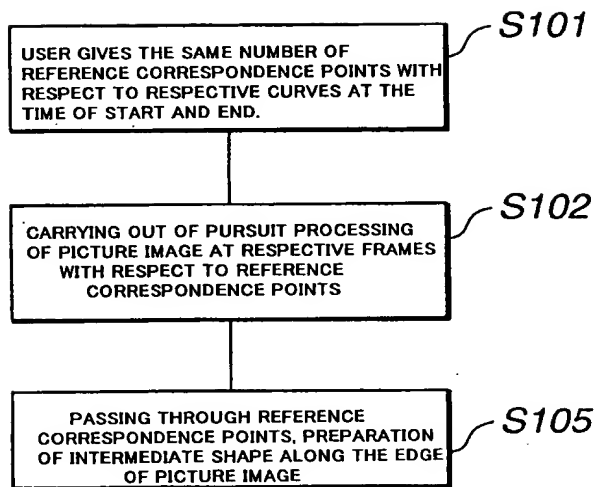


## TRANSFORMING INTO BEZIER CURVE

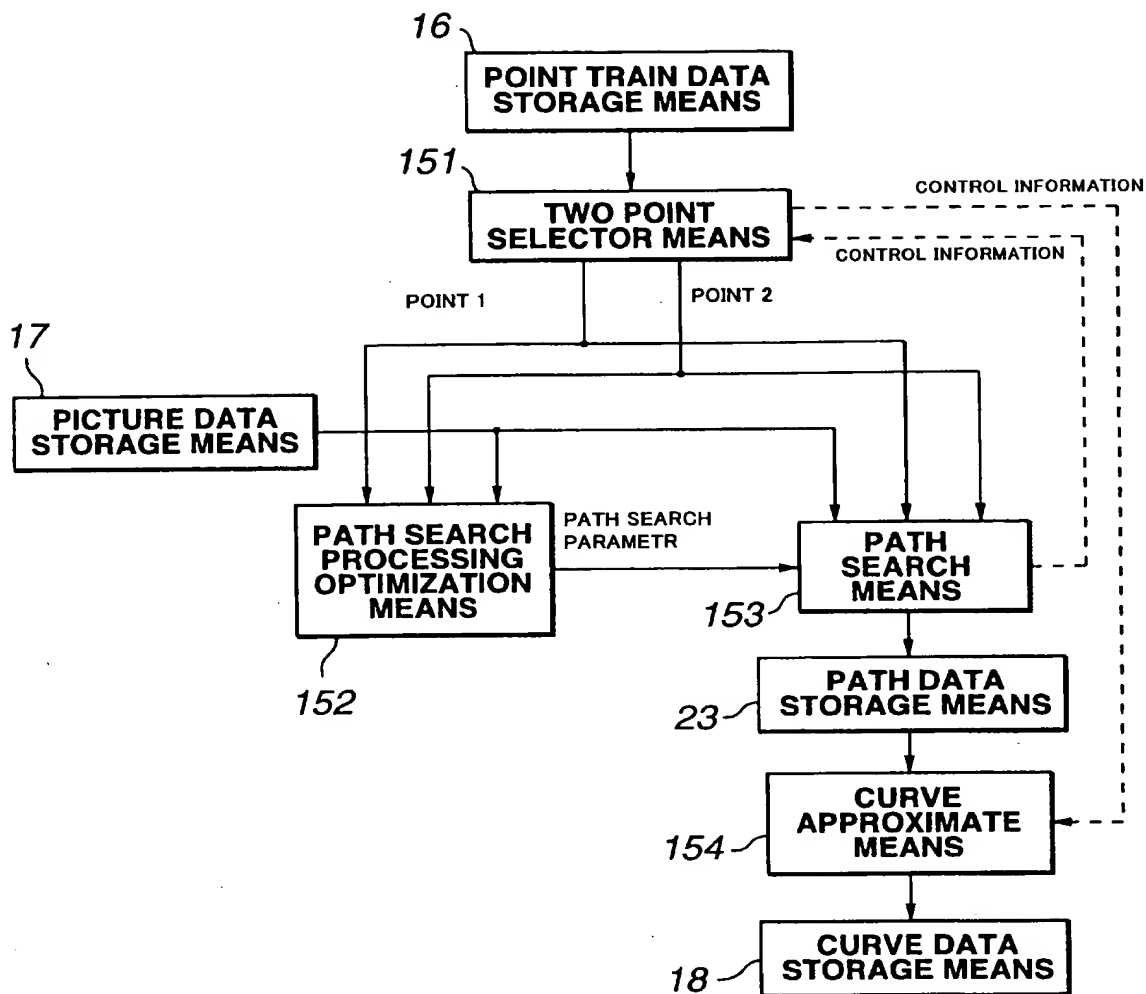
**END**



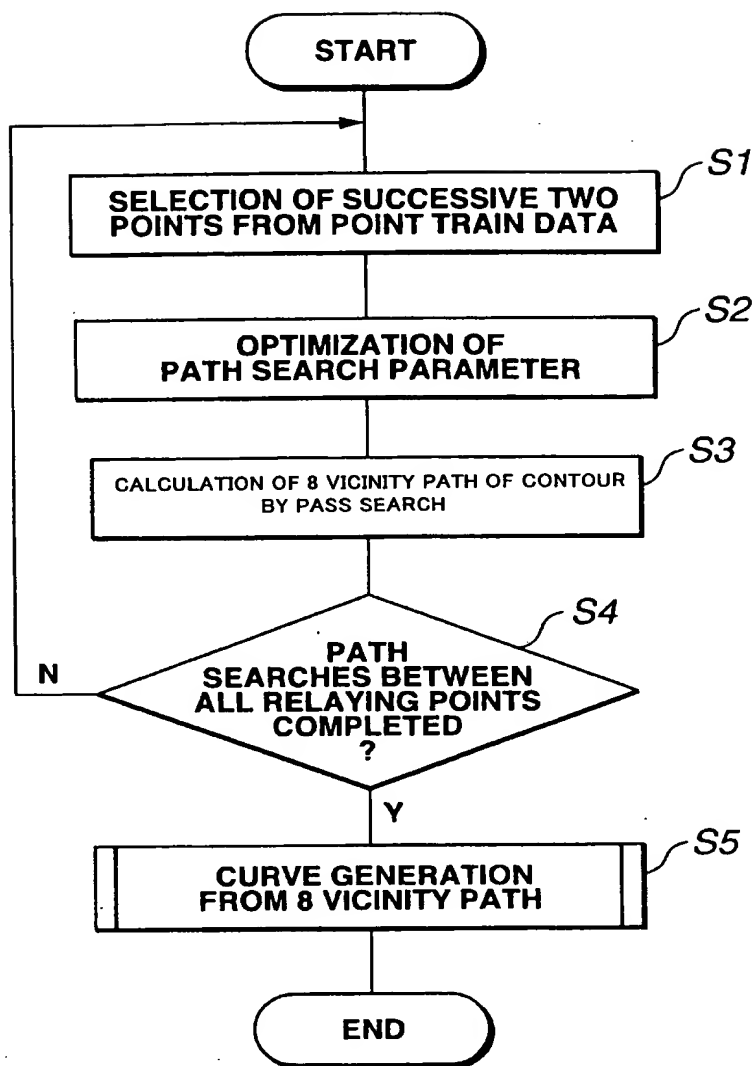
### RESULT OF TRANSFORMING WITH RESPECT TO ALL SECTIONS



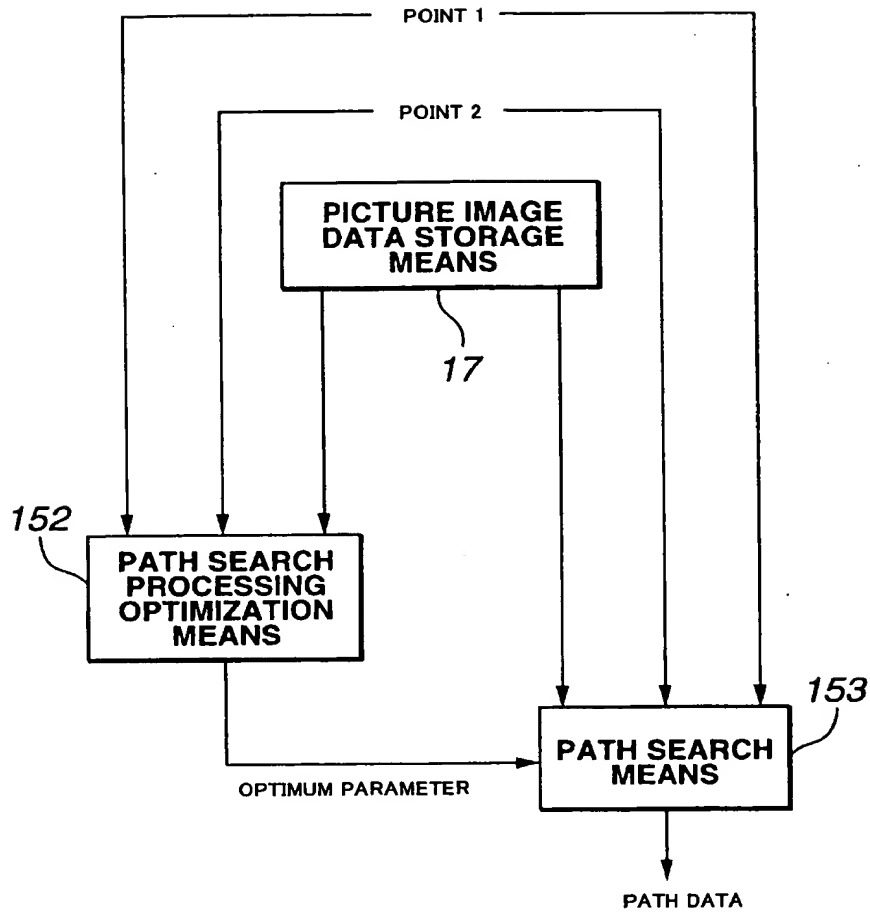
[FIG. 17]



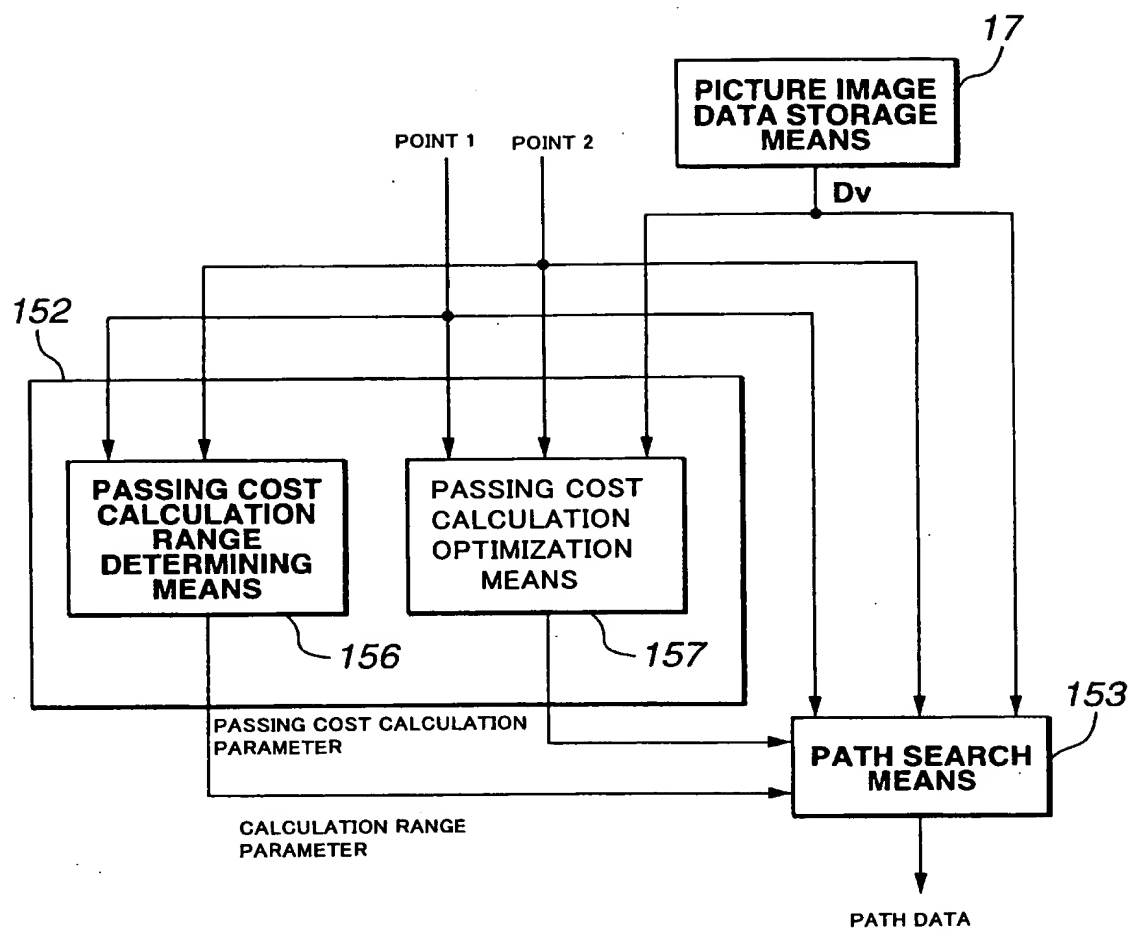
[FIG. 18]



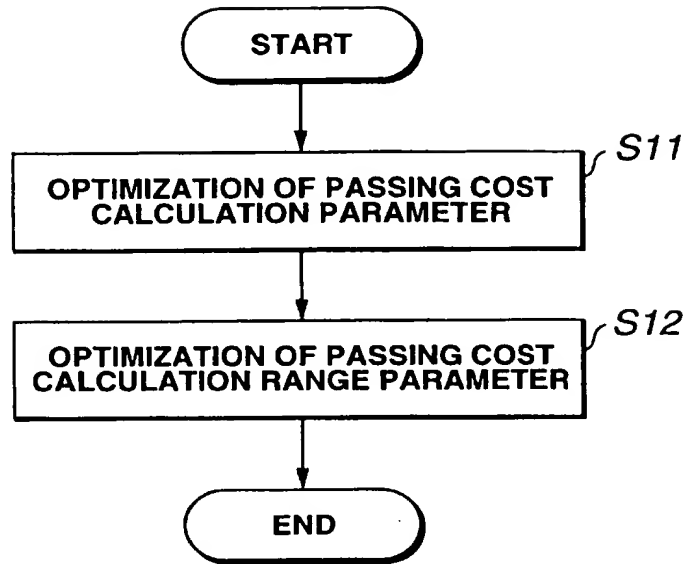
[FIG. 19]



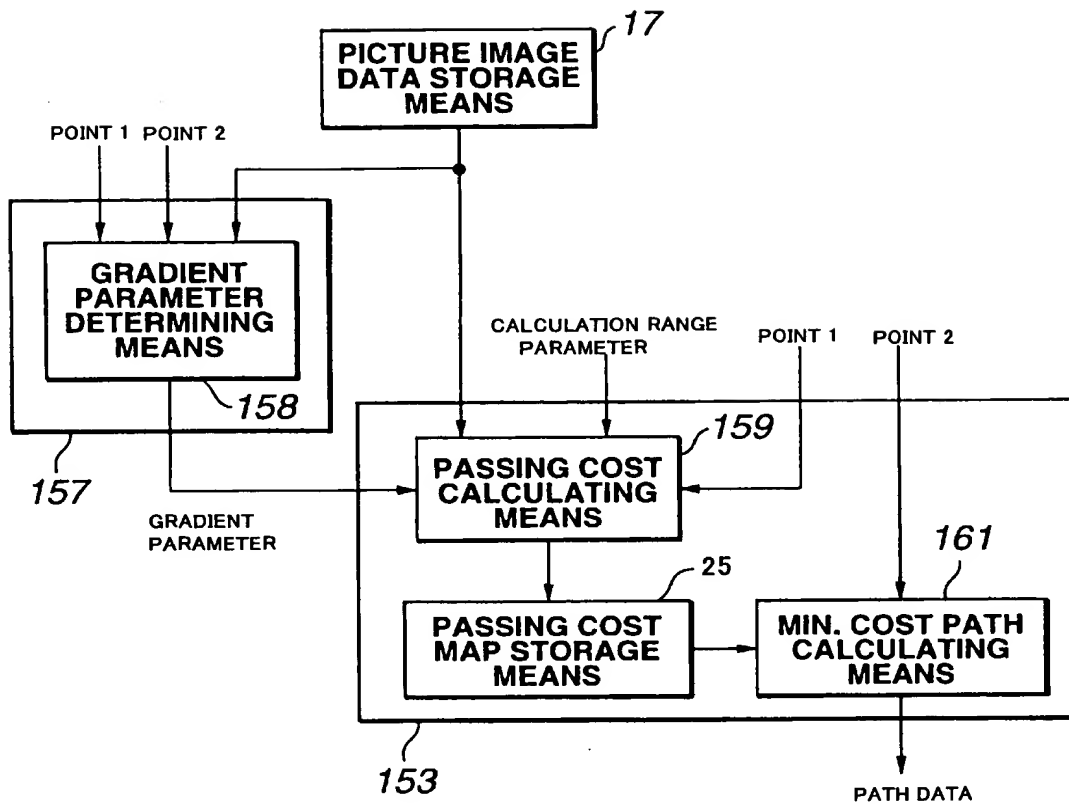
[FIG. 20]



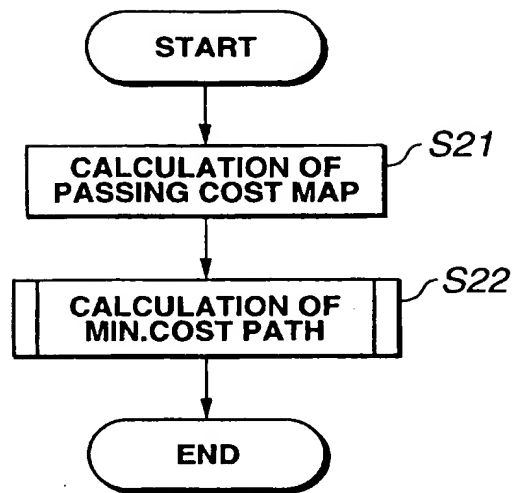
[FIG. 21]



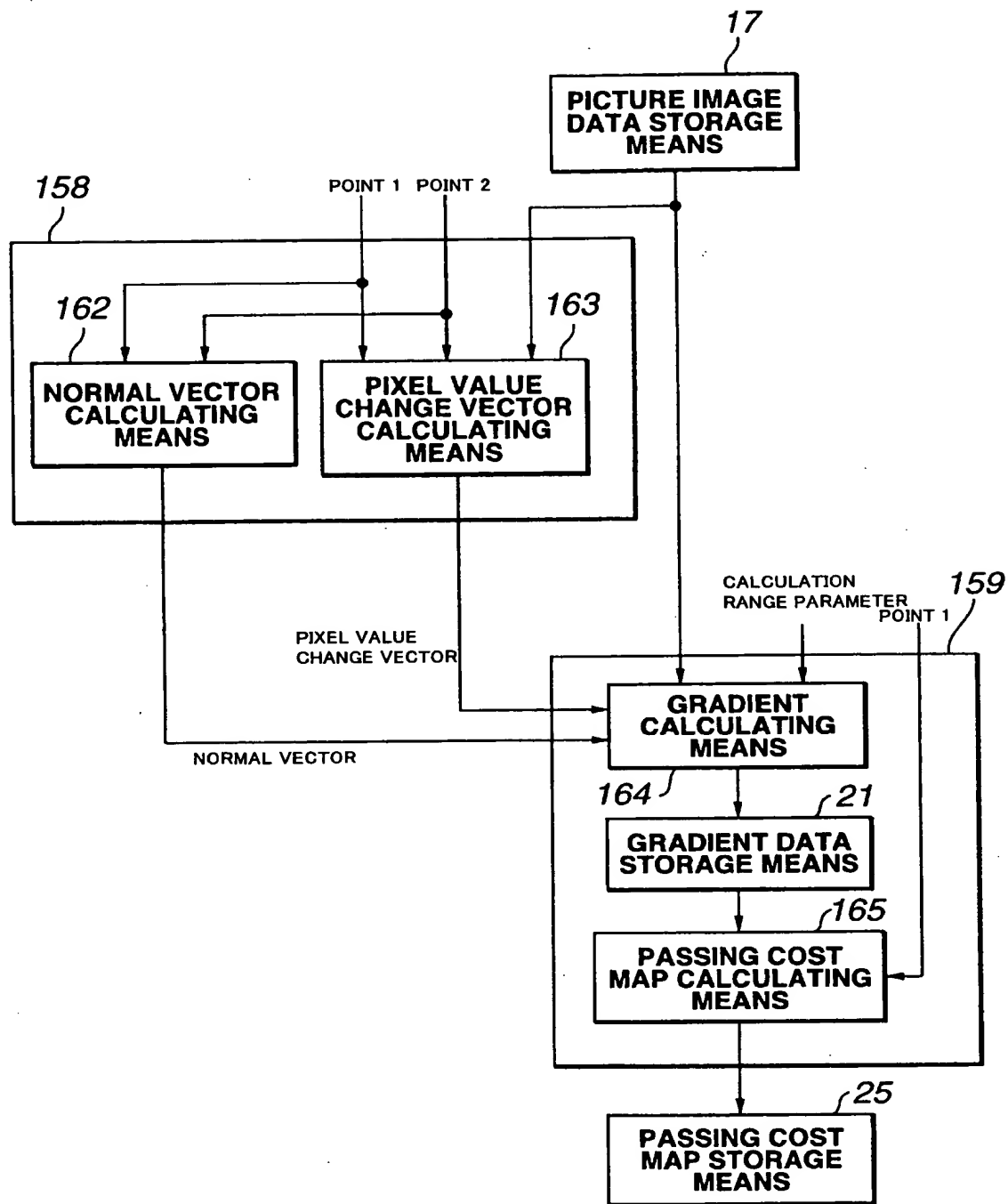
[FIG. 22]

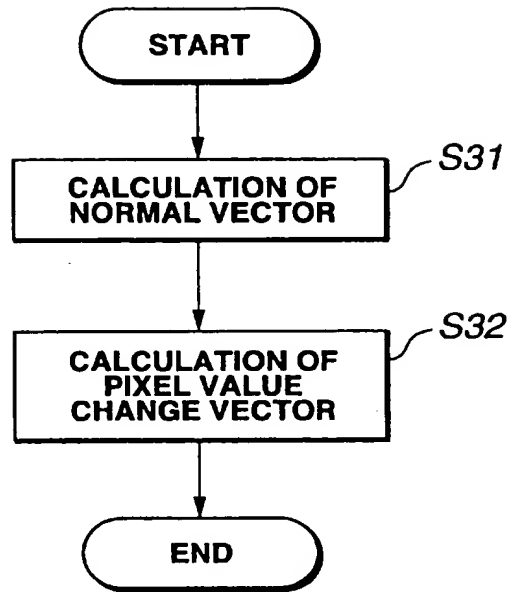


[FIG. 23]



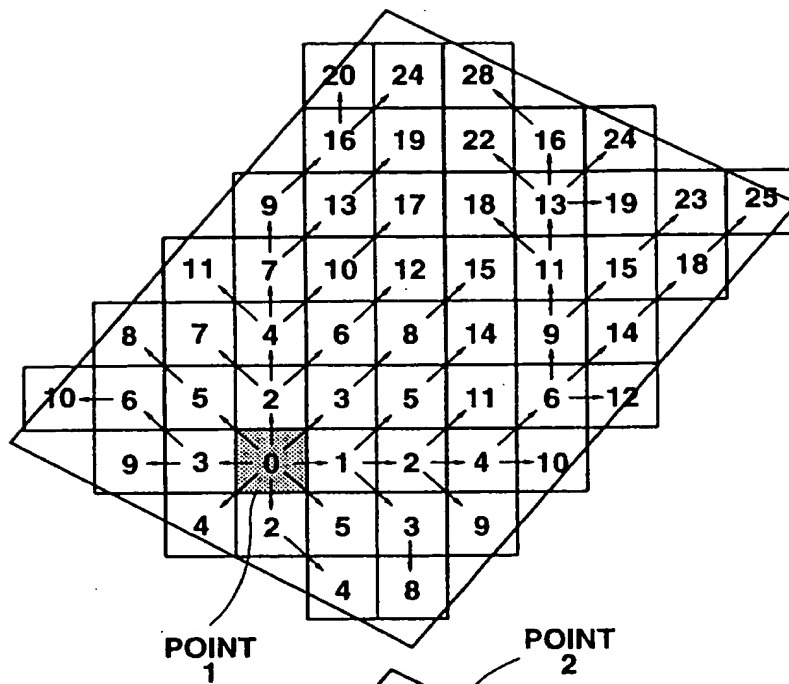
[FIG. 24]



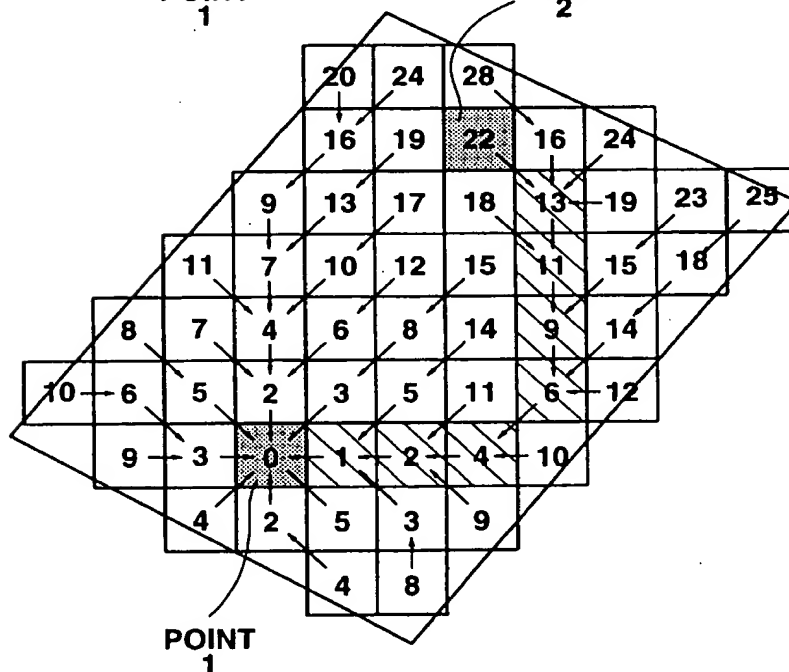


[FIG. 26]

(A)



(B)



[illegible]

**Input:**

## Data Structures:

**Output:**

### Algorithm:

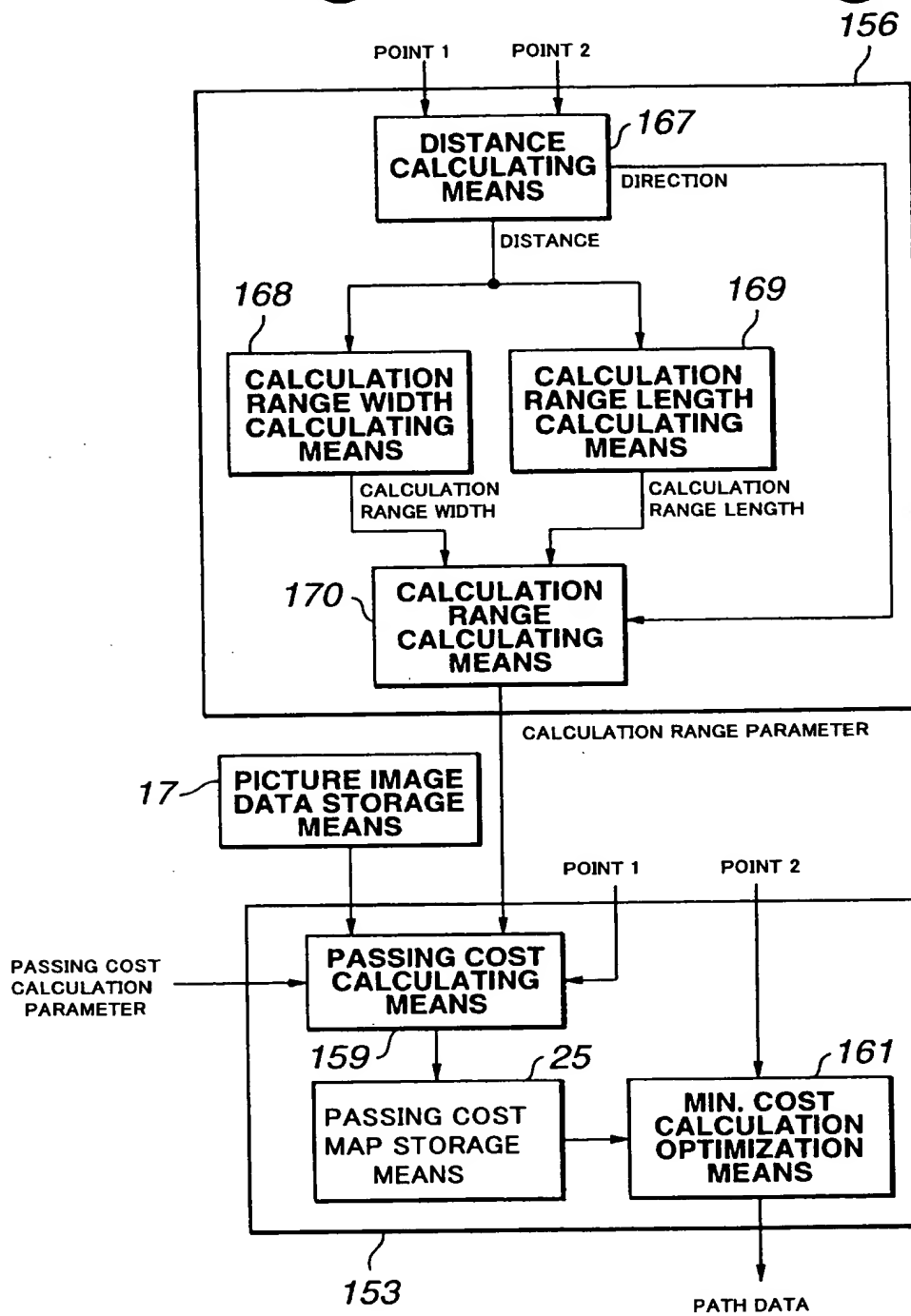
```

g(s)=0; L=s;
while L!=NULL do begin
    q=min(L)
    e(q)=TRUE;
    for each r∈N(q) such that not e(r) do begin
        gtmp=g(q)+l(q,r);
        if r∈L and gtmp < g(r) then
            r=L;
        if !(r∈L) then begin
            g(r)=gtmp;
            p(r)=q;
            L=r;
        end
    end
end
end

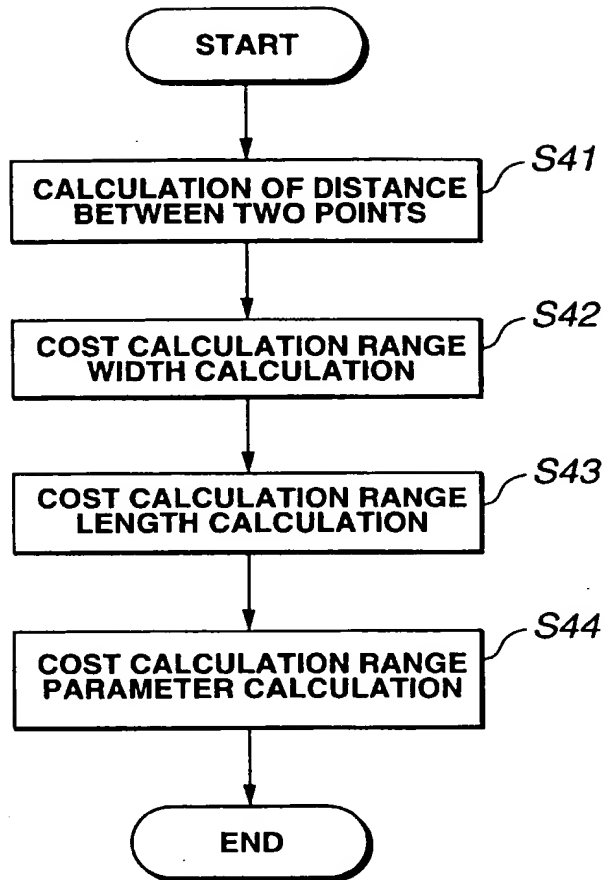
```

{Initialize active list with zero cost seed pixel.}  
 {While still points to expand:}  
 {Remove minimum cost pixel q from active list.}  
 {Mark q as expanded(i.e.,processed).}  
 {Compute total cost to neighbor.}  
 {Remove higher cost neighbor's}  
 { from list}  
 {If neighbor not on list,}  
 { assign neighbor's total cost,}  
 { set (or reset) back pointer,}  
 { and place on (or return to))  
 { active list.}

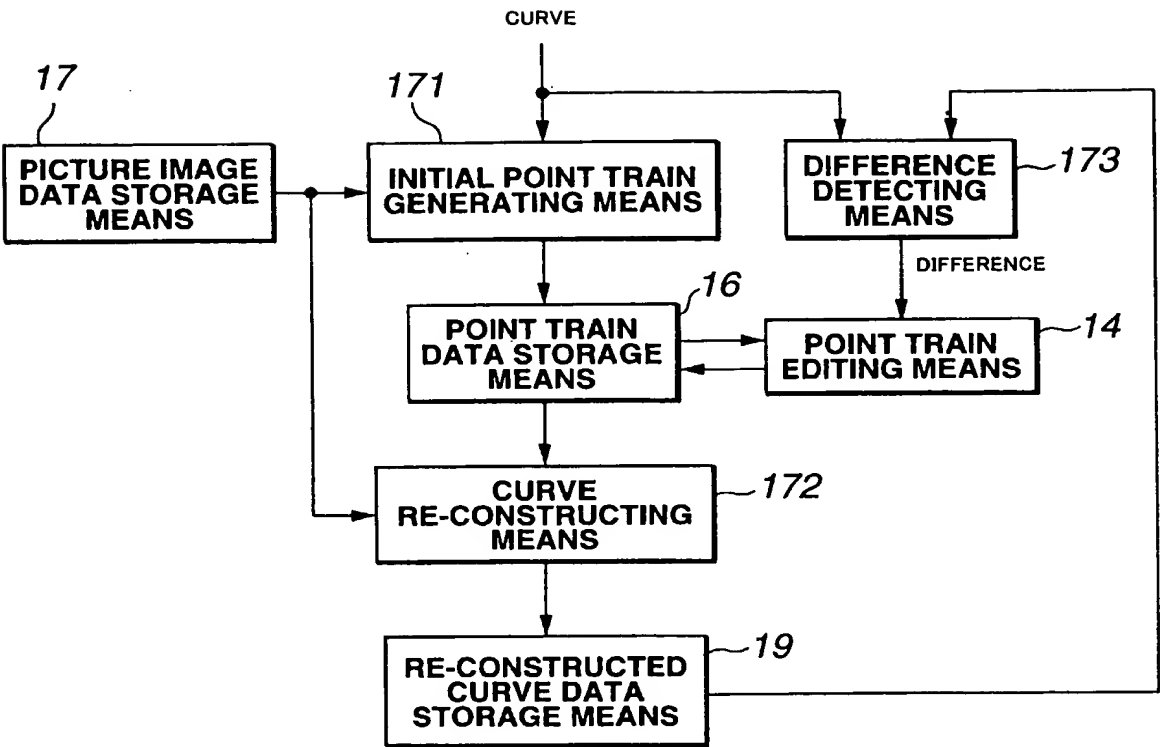
[FIG. 28]



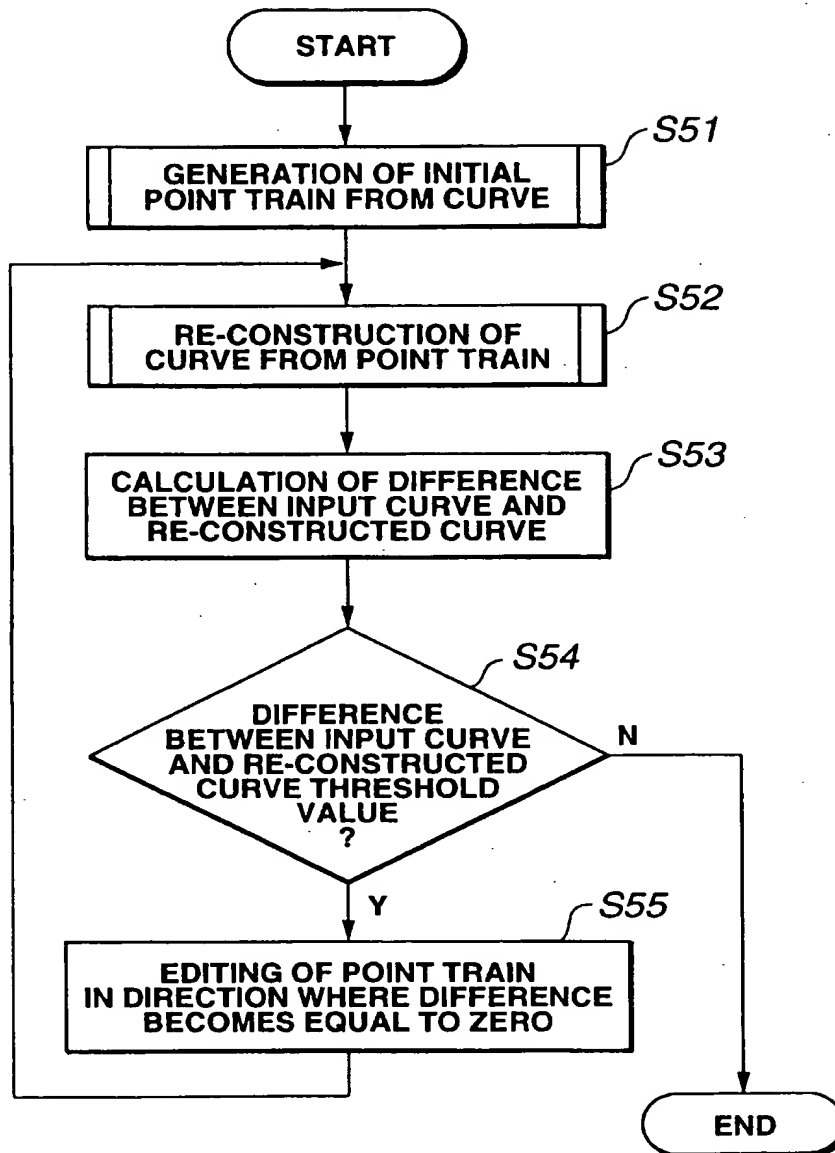
[FIG. 29]



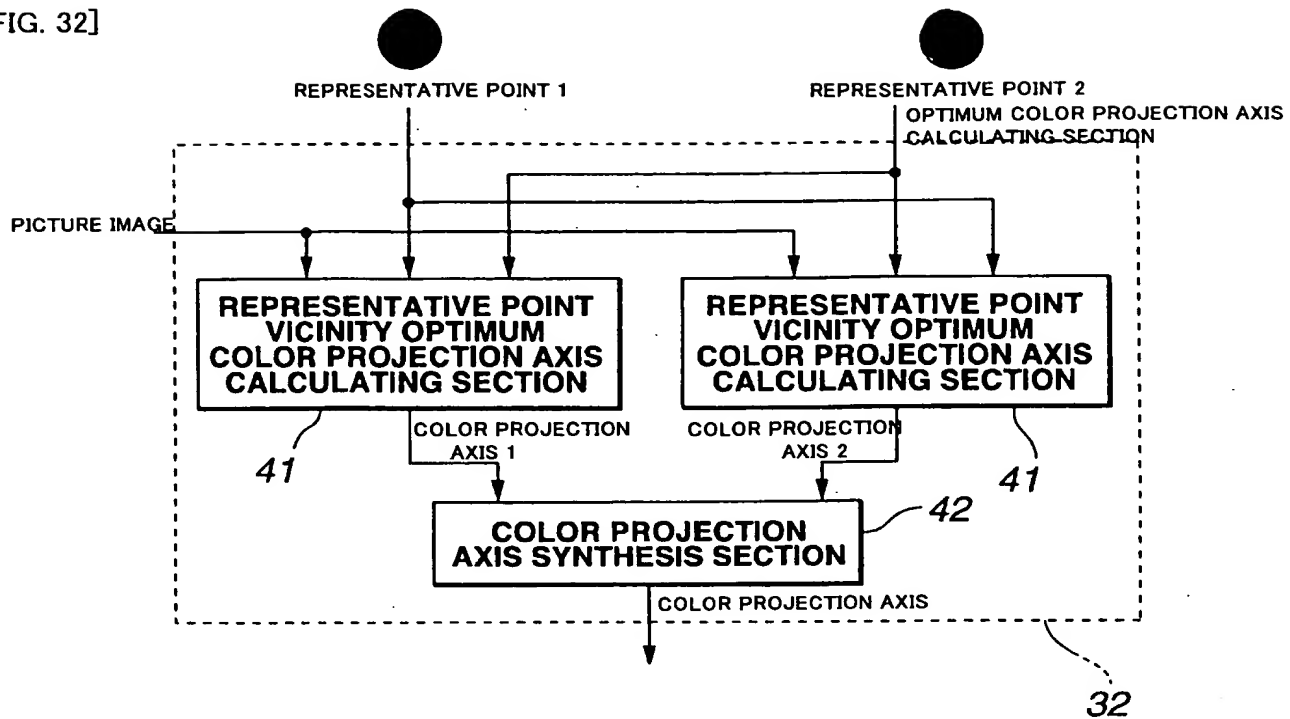
[FIG. 30]



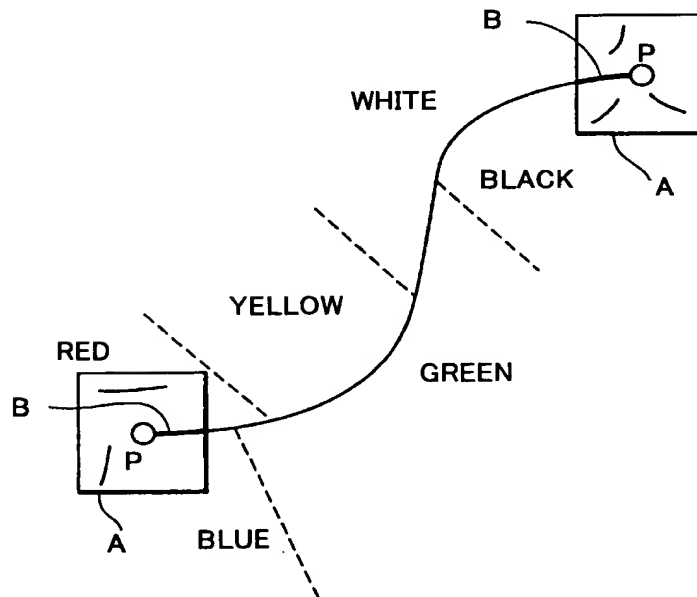
[FIG. 31]



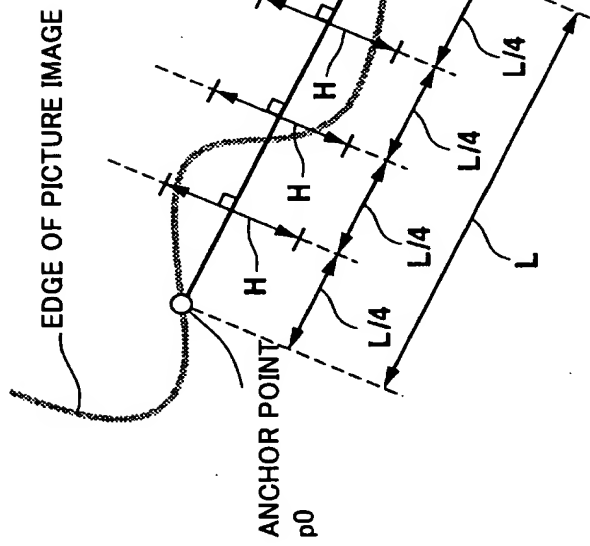
[FIG. 32]



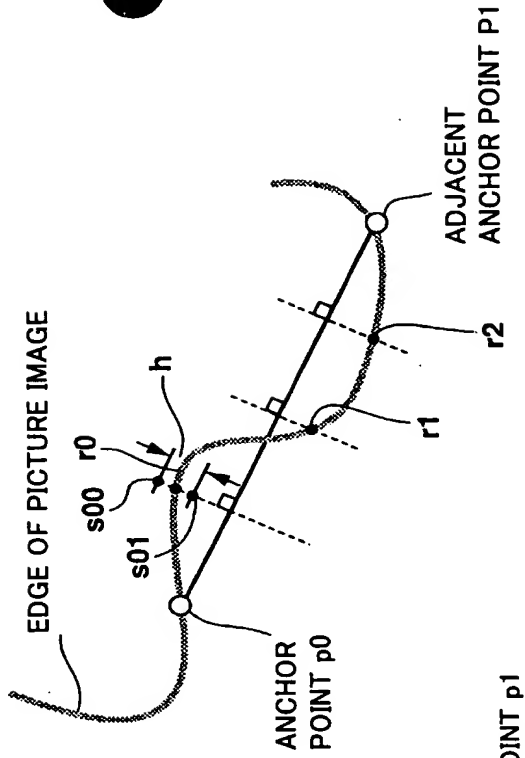
[FIG. 33]



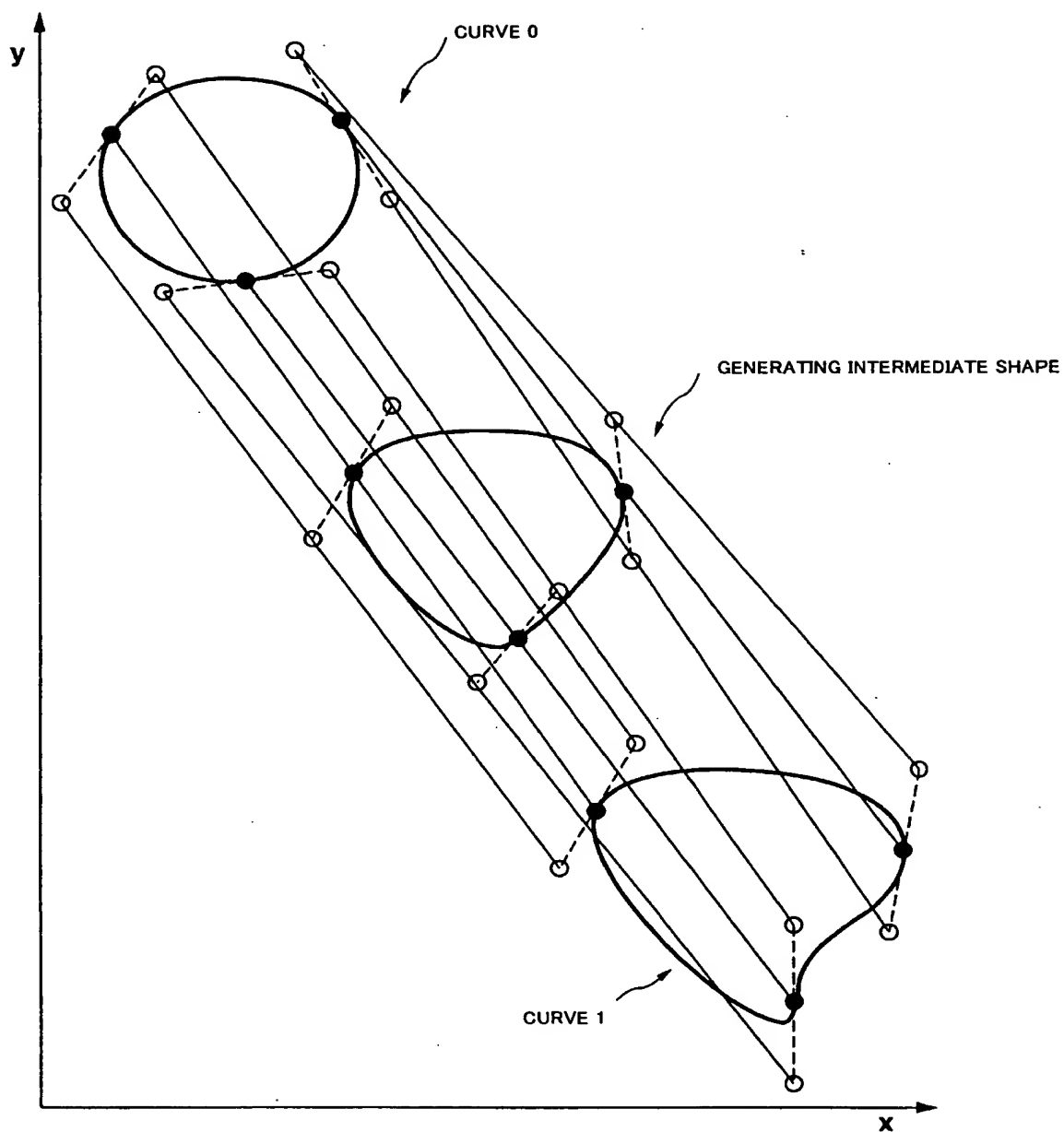
[FIG. 34]



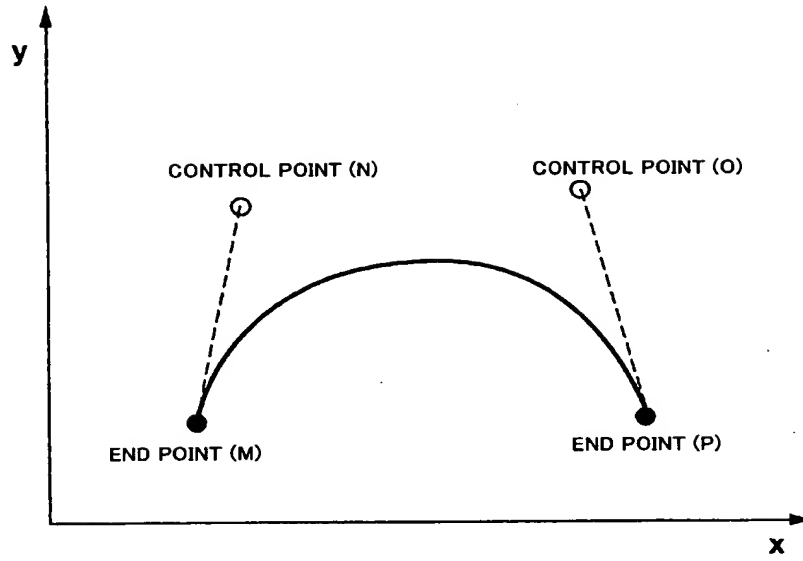
(A)



(B)



[FIG. 36]



[FIG. 37]

